Publishing Guide

How to publish data into an ESG data portal.

This describes the process of publishing data into an ESG data portal, such as the IPCC AR4 Model Output portal and the CCES/C−LAMP data portal. 'Publishing' data is the process of making data visible to the ESG portal, by moving it to the correct location and adding related information into the ESG metadata database.

Notable features of these portals:

- Data is organized one variable per file. Data for a given variable may be split across multiple files, along the time dimension.
- A relational status database tracks the status of published files. All publishing scripts coordinate with this database.
- Search capabilities in the portal support search for specific variables. The distinguishing characteristics of a dataset are model name, experiment (scenario), temporal frequency, submodel (e.g., atmosphere, ice), and run number (within an ensemble).

Overview

The steps to publish data into an ESG portal are:

1. Generate data that follows the IPCC AR4 Model Output data requirements. One way is to filter data through the CMOR library. (Note: The publishing scripts do not require that variables employ the IPCC variable names or table structures. However, it is assumed that all variables for a given table have the same temporal frequency and submodel.)
2. Configure the publishing system (at system setup).
3. Set the environment.
4. Run quality control checks.
5. Move the data to the correct repository location.
6. Publish the XML descriptions to the portal.
7. Manage existing data.
8. Unpublish data if necessary.

Database Structure

The ESG metadata database is structured as a hierarchy of objects:

```
Â Â Â Â  Project
Â Â Â Â Â Â  |
Â Â Â Â  Ensembles
Â Â Â Â Â Â  |
Â Â Â Â  Simulations
Â Â Â Â Â Â  |
```

Â Â Â Â Datasets

Projects, ensembles, and simulations are all examples of *activities* that generate data. These terms and others are defined as follows:

Â activity	A project, ensemble, or simulation.
Â dataset	A collection of related files, typically belonging to one run.
Â ensemble	A collection of simulations run by one model, for one experiment. For example: 'the CCSM Climate of the 20th Century'.
Â experiment	A scenario, such as 'Climate of the 20th Century'.
Â logical file	A data file, that may be replicated at one or more physical locations.
Â parameter	The name of a variable, independent of a particular file For example: 'surface_temperature'.
Â project	E.g., the IPCC 4th Assessment, or C−LAMP.
Â run	Same as 'simulation'.
Â simulation	A single run of a GCM. Runs are identified numerically, starting with '1'.

Database Identifiers

Â Â Each activity and dataset has a unique identifier:

- Project: There is only one top−level project, with ID 'pcmdi.<project>'.
  For example: 'pcmdi.c−lamp'
- Ensemble: IDs have the form <project>.<model>.<experiment>
  For example: 'pcmdi.ipcc4.ccsm.20c3m'
- Simulation: <ensemble>.run<run>, e.g.:
  'pcmdi.ipcc4.ccsm.20c3m.run2'
- Dataset:Â <simulation>.data, e.g.:
  'pcmdi.ipcc4.ccsm.20c3m.run2.data'
- Logical File: <dataset>.<filename>, e.g.:
  'pcmdi.ipcc4.ccsm.20c3m.run2.data.cl_A1.nc'

Directories

<ESGHOME> − ESG publishing directory
<GLOBUSHOME> − Globus middleware installation directory
<JAVAHOME> − Java installation directory
<PYTHONHOME> − Python installation directory

CMOR

CMOR is a software I/O library that writes data that complies with the IPCC AR4 Model output data requirements.

Libraries needed to build CMOR applications:

- CMOR library
- cdunif library
- udunits library

Environment

The publishing scripts are written in Python. Some of the scripts run other programs written in Java. The Python scripts are installed in the Python bin directory. To ensure that they are available, add this directory to the path. For example, in csh:

```
% set path = (<PYTHONHOME>/bin $path)
```

where <PYTHONHOME> is the python installation directory.

To setup the Java routines (in csh):

```
% setenv JAVA_HOME <JAVAHOME>
% source <ESGHOME>/bin/esg_publishing_setup.csh
```

Finally, the script *publishesg.py* accesses the ESG Replica Location Server (RLS) using a Globus proxy certificate, generated by grid−proxy−init. This requires having an ESG certificate, obtainable from the DOEgrids Certificate Manager. Select the Affiliation 'ESG'.

```
% set path = (<GLOBUSHOME>/bin $path)
% grid-proxy-init
```

Quality control

At this point it is assumed that the data is on scratch disk space, waiting to be copied to the correct data repository location. It is also assumed that the data is in netCDF format, contains one data variable per file, and complies with the CF−1/IPCC data conventions. If the data is filtered through the CMOR library, it will comply with the required conventions.

Before copying data files from scratch storage to the data repository, it is important to check the file metadata, to ensure that it correctly identifies the file. The quality control scripts check the values of three global attributes used for file identification, and correct them if necessary:

- table_id : Identifier of the table containing the variable. This attribute is used to determine the submodel and temporal frequency.
- experiment_id : Identifier of the experiment / scenario.
- realization : Run number, starting with 1.

The steps are:

1. Run *prefix.py* to generate a text summary of the key metadata, listed by subdirectory.
2. Check the summary and edit if necessary.
3. Run *fixreal.py* to ensure that all files match the summary.

For example, suppose there are two subdirectories containing the data to be published (relative to the current directory): 20C3M/A2a/run1 and 20C3m/A2a/run2. Running:

```
% prefix.py -x fix_mapfile.txt 20C3M
```

generates fix_mapfile.txt containing:

```
/scratch/20C3M/A2a/run1 1 20c3m A2 hfls_A2a_1961-1970_20C3M_run1.nc
/scratch/20C3M/A2a/run2 2 20c3m A2 hfls_A2a_1961-1970_20C3M_run2.nc
```

where the format of each line is:

```
directory run experiment table sample_file
```

Note that *prefix.py* does not scan every file but rather one file from each leaf directory. The assumption is that each leaf directory contains files having the same table, experiment, and run.

The summary file looks correct (it can be edited if necessary at this point) so fixreal.py is run to scan all files, ensuring that they match the summary. All changes are echoed. In this example one file is found with an incorrect table ID, which is corrected.

```
% fixreal.py -n fix_mapfile.txt .
./20C3M/A2a/run2/tasmin_A2a_1971-1980_20C3M_run2.nc:table_id was A1, => A2
%
```

Moving data

Once the data is quality controlled, it can be moved to the correct location in the data repository and registered in the status database, using *movefiles.py*:

```
% movefiles.py -m CASA -d .
Copy ./npp_A2.nc to /ESG2/data1/ftp/C-LAMP/i01.01/CASA/run1/atm/MA/npp/npp_A2.nc
SQL: replace into file values ("npp_A2.nc", "i01.01", "run1", "CASA", "npp", "T1", "processed", N
...
%
```

movefiles.py has a number of options. The most commonly used are:

```
Â Â Â  -dÂ Â Â Â Â Â Â Â Â Â Â Â  : Add an entry in the status database for this file.
    -fÂ Â Â Â Â Â Â Â Â Â Â Â  : Force - overwrite existing files.
    -h Â Â  Â Â Â Â Â Â Â  : print a help message
    -iÂ Â Â Â Â Â Â Â Â Â Â Â  : Ignore 'File exists' errors, keep processing
    -m <model>    : Model acronym
```

Scanning and publishing data

Once the data is in place it can be published into the portal catalogs. The steps are:

- Generate a list of datasets to be scanned, using *listmods.py*.
- Foreach dataset:
  Scan the files using *genxml.py*.
  Publish using *publishesg.py*.

Steps 2. and 3. can be combined using *rungen.py*, which in turn runs genxml.py and publishesg.py for each dataset.

For example, suppose a set of files was just published for model CASA. *listmods.py* can be used to create a summary listing of the datasets associated with those files, by default for those copied on the current day:

```
% listmods.py > <ESGHOME>/publishing/CASA/process_1.txt
```

generating

```
| CASA | i01.01 | run1 | T1 | 10 |
```

This indicates that 10 files were registered for the dataset where model=CASA, experiment=i01.01, realization=run1, and table=T1.
To scan and publish the data, run *rungen.py* which takes as input the summary listing generated by *listmods.py*.

```
% rungen.py --password <password> -t\| <ESGHOME>/publishing/CASA/process_1.txt
genxml.py --justdoit --verbose --experiment i01.01 --frequency monthly --genpath --model CASA --r
...
Wrote files:
Â  <ESGHOME>/publishing/CASA/pcmdi.c-lamp.CASA.i01.01.run1.monthly.xml (Dataset/Simulation descri
Â  <ESGHOME>/publishing/CASA/pcmdi.c-lamp.CASA.i01.01.xml (Ensemble description)
  <ESGHOME>/publishing/CASA/C-LAMP.pcmdi.c-lamp.CASA.i01.01.run1.pars (Configuration file)

publishesg.py --verbose --password <password> <ESGHOME>/CASA/pcmdi.c-lamp.CASA.i01.01.run1.monthl
...
%
```

Note that *genxml.py* generates a set of files in ESGML markup format, which are subsequently processed by *publishesg.py*.

Also note that when files are scanned by *genxml.py*, the time axis is checked for monotonicity. If this check fails then *genxml.py* will fail for that dataset. (Look for zero−length .xml files in the output directory, and examine the output log to find the file that failed the scan.) The script *checkTimes.py* is helpful in determining exactly where the error occurred.

Manage existing data

Once data has been published, it may still be necessary to withdraw the data, if errors are found. Similarly a file or set of files may need to be renamed or deleted. Several scripts are provided to coordinate these tasks with the status database:

deletefile.py	Delete a file or set of files, and delete the corresponding database entries. See *markasbad.py*
deletevar.py	Delete the file or files associated with a particular variable, and delete the corresponding database entrie
listfiles.py	List all files that match a given criteria. IMPORTANT: The database table *param* must have an entry fo contained in the files, otherwise they will not be listed.
markasbad.py	Withdraw a file or set of files. In contrast to deletefile.py, the files are renamed, world−read mode remo database entries are set to 'withdrawn' rather than being removed.
renamefile.py	Rename a file or set of files, and update the corresponding status database entries.

Unpublishing data

If an error is made in publishing data, the metadata can be removed using the *unpublishesg.py* script. Unlike the publishing script, *unpublishesg.py* takes an activity or dataset id as input. The general usage of the script is:

```
unpublishesg.py [options] <objid> [<objid> ...]
```

where <objid> is an activity, dataset, logicalFile, or parameter identifier as defined above.

Note: the objid for most objects can be found by selecting the metadata link in the portal.

By default, the object is deleted from all metadata subsystems: the RLS, relational database, and THREDDS catalogs. For example, deletion of a dataset removes all related files from the RLS, all associated entries from the relational database, and removes the link from the parent THREDDS catalog so that the dataset is no longer visible in the web portal.

Note that when THREDDS catalogs are updated, the −−parsfile option isÂ needed to specify a .pars configuration file. This file is generated by the publishing script.

Examples

- Unpublish metadata for run 1 of the CCSM Climate of the 20th Century ensemble:

```
Â Â  % unpublishesg.py --recurse --parsfile IPCC.pcmdi.ipcc4.ccsm.20c3m.run1.pars pcmdi.ipcc4.ccs
```

- Unpublish the ensemble, simulations, and datasets for the CCSM Climate of the 20th Century:

```
Â Â  % unpublishesg.py --verbose --recurse --parsfile IPCC.pcmdi.ipcc4.ccsm.20c3m.run1.pars pcmdi
```

Â Â Note that the .pars files for any of the runs in the ensemble may be used to unpublish the ensemble.

- Unpublish the dataset associated with run1:

```
Â Â  % unpublishesg.py --verbose --parsfile IPCC.pcmdi.ipcc4.ccsm.20c3m.run1.pars pcmdi.ipcc4.ccs
```

- Â Unpublish a logical file:

```
Â Â  % unpublishesg.py --verbose --parsfile IPCC.pcmdi.ipcc4.ccsm.20c3m.run1.pars pcmdi.ipcc4.ccs
```

- Remove the RLS entries for a dataset:

(NOTE: THIS SHOULD BE DONE BEFORE PUBLISHING, IF FILES HAVE BEEN PERMANENTLY RENAMED OR REMOVED FROM THE ARCHIVE!)

```
Â Â  % unpublishesg.py --verbose --password <password> --rls pcmdi.ipcc4.cnrm_cm3.20c3m.run1.mont
```